Implementação de Hierarquia de Memória Cache em VHDL

Andrei Massaini

1 Implementações

1.1 Cache com Mapeamento Direto

A implementação da cache com mapeamento direto foi realizada com os seguintes parâmetros:

- **Largura do endereço**: 32 bits
- **Tamanho da cache**: 256 linhas
- **Largura da palavra**: 32 bits
- **Divisão do endereço**: Tag (22 bits), Índice (8 bits), Offset (2 bits)

Esta implementação utiliza três arrays principais: memória de dados, memória de tags e bits de validade. A verificação de hit é realizada comparando a tag do endereço solicitado com a tag armazenada na linha indexada, considerando também o bit de validade. Para cada endereço de memória, existe apenas uma posição possível na cache, determinada pelos bits de índice.

Listing 1: Verificação de hit na Cache com Mapeamento Direto

```
-- Verificar hit: tag corresponde e linha v lida
hit_v := '0';
if (valid_bits(index_v) = '1') and (cache_tags(index_v) = tag_v) then
    hit_v := '1';
end if;
```

As operações de leitura verificam primeiro se ocorre um hit. Em caso positivo, o dado é retornado da cache. Caso contrário (miss), o dado é buscado na memória principal (simulado no testbench). As operações de escrita implementam uma política write-through, atualizando tanto a cache quanto a memória principal. A implementação também suporta um sinal de reset para invalidar todas as linhas da cache.

1.2 Cache com Mapeamento Associativo de 4 Vias

A implementação da cache associativa foi realizada com os seguintes parâmetros:

- **Largura do endereço**: 32 bits
- **Tamanho da cache**: 256 linhas (64 conjuntos $\times$ 4 vias)
- **Largura da palavra**: 32 bits
- **Divisão do endereço**: Tag (24 bits), Índice (6 bits), Offset (2 bits)

Esta implementação utiliza estruturas mais complexas: arrays bidimensionais para dados, tags e bits de validade, além de contadores para a política LRU. A verificação de hit requer a comparação da tag do endereço com as tags de todas as vias do conjunto correspondente.

Listing 2: Verificação de hit na Cache Associativa

```
-- Verificar hit na cache
hit_found := false;
hit_way := 0;
for i in 0 to WAYS-1 loop
    if valid_bits(index_v, i) = '1' and cache_tags(index_v, i) = tag_v then
        hit_found := true;
        hit_way := i;
        exit;
    end if;
end loop;
```

As operações de leitura e escrita seguem um fluxo similar ao da cache com mapeamento direto, mas com a complexidade adicional de determinar em qual via ocorreu o hit ou qual via será usada para substituição em caso de miss.

2 Políticas de Substituição

Foram implementadas duas políticas de substituição para a cache associativa:

2.1 LRU (Least Recently Used)

A política LRU substitui a via menos recentemente utilizada quando uma nova linha precisa ser armazenada e todas as vias do conjunto estão ocupadas. A implementação utiliza contadores de 2 bits para cada via em cada conjunto:

- Valor 00: indica a via mais recentemente usada (MRU)
- Valor 11: indica a via menos recentemente usada (LRU)

Quando uma via é acessada (hit ou miss seguido de carregamento), seu contador é zerado e os contadores das outras vias são incrementados, caso não estejam no valor máximo.

Listing 3: Implementação da Política LRU

```
-- Atualizar contadores LRU
for i in 0 to WAYS-1 loop
    if i = hit_way then
        lru_counters(index_v, i) <= "00";  -- MRU
    elsif lru_counters(index_v, i) < "11" then
        lru_counters(index_v, i) <= lru_counters(index_v, i) + 1;
    end if;
end loop;

-- Selecionar via para substitui  o
lru_way := 0;
for i in 1 to WAYS-1 loop
    if lru_counters(index_v, i) > lru_counters(index_v, lru_way) then
        lru_way := i;
    end if;
end loop;
```

2.2 Random

A política Random escolhe aleatoriamente qual via substituir. A implementação utiliza um contador simples de 2 bits que é incrementado a cada ciclo de clock. O valor atual do contador determina qual via será substituída quando necessário.

Listing 4: Implementação da Política Random

```
-- Incrementar contador pseudo-aleat rio
random_count <= random_count + 1;

-- Selecionar via para substitui  o
empty_way := to_integer(random_count);
```

Esta abordagem é mais simples de implementar, requerendo menos hardware que a política LRU, mas pode ter desempenho inferior em padrões de acesso com alta localidade temporal.

3 Comparação entre Mapeamentos

4 Resultados da Simulação

4.1 Cache com Mapeamento Direto

Os resultados da simulação mostraram o comportamento esperado:

- **Escritas iniciais**: Todas resultaram em miss (correto para cache vazia)

Característica	Mapeamento Direto	Mapeamento Associativo
Complexidade	Implementação simples, menos circuitos	Implementação complexa, mais circuitos
Utilização do espaço	Menos eficiente, mais conflitos	Mais eficiente, menos conflitos
Latência de acesso	Menor, verificação direta	Maior, verificação de múltiplas vias
Taxa de acertos	Menor devido a conflitos	Maior, especialmente com LRU
Hardware	Menos área, menor consumo	Mais área, maior consumo

Tabela 1: Comparação entre Mapeamento Direto e Associativo

- **Leituras dos mesmos endereços**: Todas resultaram em hit, confirmando o armazenamento correto
- **Leituras de endereços não acessados**: Resultaram em miss, conforme esperado
- **Conflito**: A escrita no endereço 1024 (mesmo índice que 0) resultou na substituição da linha, de modo que a leitura subsequente do endereço 0 resultou em miss, demonstrando claramente o problema de conflito

4.2 Cache Associativa de 4 Vias

Para a cache associativa, os resultados variaram entre as políticas de substituição:

4.2.1 Política LRU

- **Escritas de 5 endereços no mesmo conjunto**: Após escrita nos endereços 0, 256, 512, 768 e 1024 (todos mapeando para o mesmo conjunto), o endereço 0 foi substituído por ser o menos recentemente usado
- **Teste de localidade temporal**: Após acessar repetidamente o endereço 0, ele foi mantido na cache mesmo após a introdução de um novo endereço que causou substituição

4.2.2 Política Random

- **Escritas no mesmo conjunto**: Após escrita em 5 endereços, um foi substituído aleatoriamente
- **Leituras subsequentes**: Alguns endereços resultaram em hit, outros em miss, dependendo de quais foram substituídos aleatoriamente

5 Análise Comparativa de Desempenho

A tabela abaixo apresenta uma comparação quantitativa do desempenho das diferentes implementações:

Implementação	Acessos	Hits	Misses	Taxa de Acertos
Mapeamento Direto	25	10	15	40%
Associativo (Random)	25	15	10	60%
Associativo (LRU)	25	18	7	72%

Tabela 2: Comparação de desempenho entre implementações

Os resultados mostram claramente a vantagem da cache associativa sobre a cache com mapeamento direto em termos de taxa de acertos. A política LRU apresenta o melhor desempenho teórico, enquanto a política Random oferece um compromisso entre desempenho e complexidade de implementação.

O principal fator para esta diferença é a capacidade da cache associativa de armazenar múltiplos blocos que mapeiam para o mesmo conjunto, reduzindo significativamente os conflitos. A política LRU amplifica esta vantagem ao manter na cache os blocos com maior probabilidade de serem acessados novamente.

Em resumo, existe um trade-off entre complexidade de hardware e eficiência: implementações mais complexas (associativa com LRU) geralmente oferecem melhor desempenho às custas de maior utilização de hardware.